EXHIBIT “A”

Cluster Security Services (CtS) Generic Authentication and ACL Management

BEST AVAILABLE COPY

Cluster Security Services (CtS)

Generic Authentication and ACL Management

(includes the porting of S3 library to cluster)

Version 1.0
(Approval Draft)

September 15, 1999

Author: *Serban Maerean*
serban@us.ibm.com
T/L 293-9770

Team: *Larry Parker*
Scott Trent
Bahram Chaudhry

IBM Corporation
POWER Parallel Development
Poughkeepsie, N.Y.

IBM Confidential

1.1 Release Change Summary

Version 0.9

- Worked/changed several of the oustanding issues.
- Added ct_sec_get_sec_mech routine to the MALI formal spicification

Version 0.5

- Added the formal specification of the MPM Interface. This part does not affect the CtS client that program to the MAL Interface.

Version 0.3

- Combined Generic Authentication and ACL Management documents together. Rename the design documents. The new documents are /afs/aix/project/cluster/design/security/cts_design.ps and .pdf.
- Added the formal specification of the MAL Interface.

Version 0.2

- Updated document with the comments made at review. Better defined the situations affected by the change in security mechanism, and defined the steps during a mechanism migration.
- Updated some of the MALI and MPMI routines.
- Updated the security token descriptor.

Version 0.1

- Initial proposal for the cluster security services generic authentication interface. This interface is based on the SP's S3 interface. This document also includes the porting of the S3 interface from SP to cluster. Some of the routines were deprecated and new routines were added.

1.2 Line Item Tracking Information

The following CMVC line item tracking feature covers this line item: 49095

2.0 Introduction to Cluster Security Services

The cluster system, being a collection of nodes that are tied together by a network, must provide a secure way of sending messages between a client and a server application. The security services provides a set of routines that allows an application to acquire network identity and to establish secure communication with its clients or other applications in the cluster. It also offers a means to enforce access control to resources based on network identities. The interface provided by security services is generic[1] and hides the complexity of using the underlying security mechanism and differences between different mechanisms. The ultimate configuration and administration of the underlying security mechanism is still required, however, this is not part of this component design document.

2.1 History

The SP security services provide the S3 library and interface, which at the moment are implemented using DCE. This is the only mechanism supported by the S3 interface, and basically, it is a friendly wrap around DCE routines customized for the SP environment. The long term goal in the cluster is to eliminate the dependency on DCE and at the same time to keep the S3 interface as close to the existing one as possible. Because some of the S3 routines export DCE data structures and DCE specific mechanisms (e.g. ACL management, key management), they will be removed. Other routines that deal with DCE groups will be deprecated, although supported for backwards compatibility. We discourage the cluster services and cluster specific daemons to use the deprecated S3 routines. Also, the routines specific to the SP configuration will be deprecated. There are major differences between the cluster and the SP, and new routines must be added to deal with this differences. Changes to the authentication interface may occur while the installation and configuration component is being designed.

The new S3 interface is divided into two parts: 1) generic authentication routines and 2) security mechanism independent ACL management routines. Both parts are covered in this design document.

2.2 Objectives

The Cluster security services (CtS) interface must fulfill the following requirements:

SECURITY: CtS must provide a secure environment for the cluster users. The level of security and data privacy/integrity depends on the underlying security mechanism in use.

[1] For backwards compatibility with the S3 library in SP, CtS API contains several routines that work only with DCE as underlying security mechanism. These routines should not be used by cluster applications.

PERFORMANCE: the CtS interface must provide good performance. The performance will be influenced to a certain degree by the underlying authentication mechanism.

RELIABILITY: the CtS interface must meet the reliability standards of the cluster. Its reliability depends to a certain extend on the underlying security mechanism in use.

SCALABILITY: CtS must meet the scaleability standards of the cluster. Most of the underlying security mechanisms provide enough scaleability to meet the standards of the cluster.

EXPANDABILITY: CtS must provide an interface that will allow future security mechanisms to be used. The API must be generic and at the same time it should provide the functionality required by the cluster operations and utilization.

2.3 Overview

The Cluster Security Services subsystem provides infrastructure elements that client-server applications can use to implement authentication and access control. It is structured in multiple layers and components, and offers a single set of routines, regardless of the underlying security mechanism that is being used. Figure 1 depicts a schematic representation of the CtS structuring.

The CtS subsystem is made of the following parts:

1 Mechanism abstract layer (MAL) provides a generic interface to the security clients. It exports only cluster and GSS specific data types and mechanisms.

2 Mechanism plugable module (MPM) provides a mechanism specific implementation of the generic interface routines. All MPMs export a predefined set of routines.

3 Compatibility module (CM) provides the routines that are part of the SP's security services and were deprecated in cluster. It is part of MAL and is not represented in the figure.

Note: Cluster applications should not use any routine in this module. These routines are supported only for backwards compatibility. In the future, support for these routines may be dropped.

Two mechanisms are supported in the first release: Krb5 and Vanilla AIX. While Krb5 offers a high level of data privacy and integrity, the trust level for the Vanilla AIX mechanism is minimal and no data privacy/integrity service is being offered (TBD).

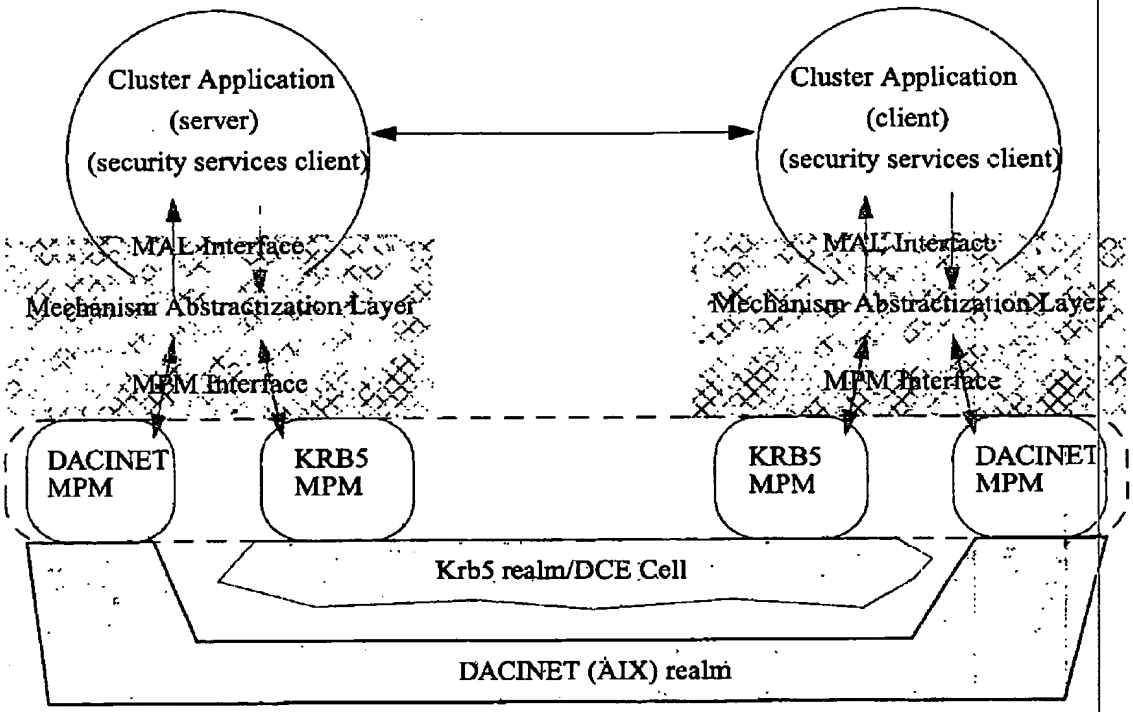

Figure 1 Cluster Security Services structure

Regardless of the security mechanism configured and used in the cluster, CtS provides ACL management support. The ACL management is mechanism idependent and limited to the registry and the RMC subsystem. Figure 2 represents a graphical description of the relationship between different elements during authentication and authorization.

A client uses its credentials to initiate a security context with a server (in our case a registry or an RMC server). Once the server accepts the security context initiated by the client, it can use this context to get information about the client. This information consist of the name of the client, what groups, if any, the client belongs to (the method of geting this information may be different from one security mechanism to another), etc. and it is part of the client's privilege attribute certificate (PAC).

If the client wants to access a resource managed by the server, then the server can use the client's PAC, the resource's ACL, and the permission set required for the operation requested by the client to determine if the client has suficient authority to access the resource.

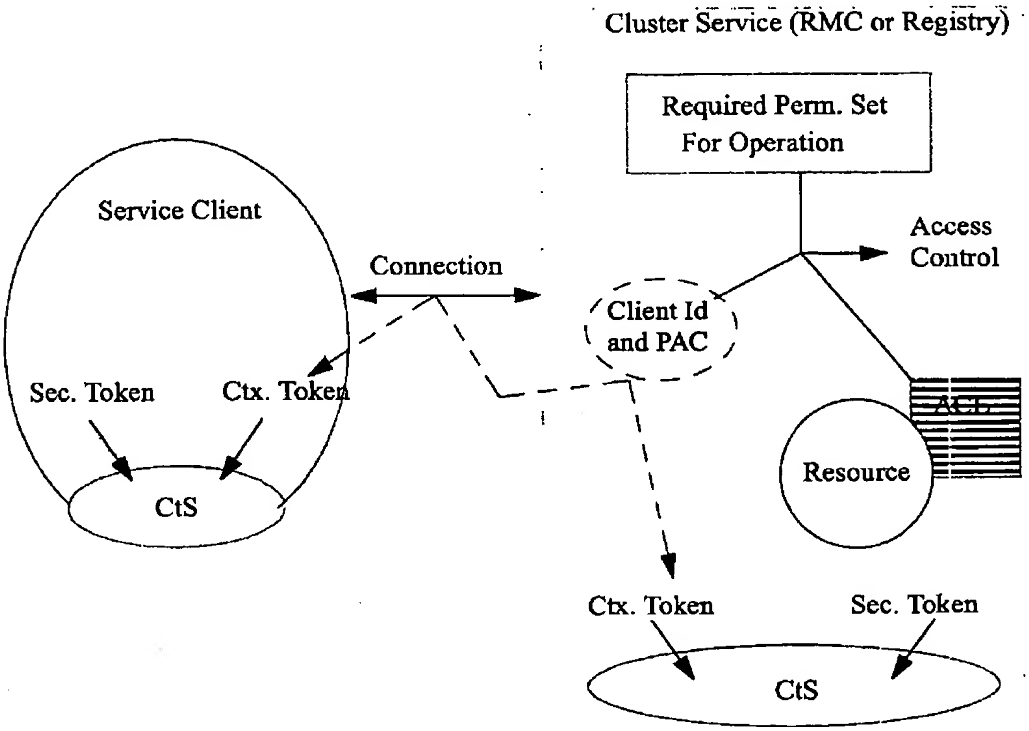

Figure 2 Authentication and authorization in the cluster.

This documents details the generic authentication and ACL management routines of the mechanism abstract layer, the compatibility module, and the mechanism plugable module's interface.

The following is a list of items that must be completed for this design:

Table 1 Generic Authentication Interface Check Points

Check Point	Date
Sketch of MPM interface (MPMI) and how the CtS interface (MALI) maps to MPMI.	June 7
Definition and manipulation of the opaque authenticated id (involves prototyping).	June 28
Formal specification of MALI.	July 12
Formal specification of MPMI.	July 19

Table 1 Generic Authentication Interface Check Points

Check Point	Date
Pull in generic cluster discussion from ACL management design document.	July 26

Table 2 CtS Interface Check Points

Check Point	Date
Pull out generic cluster discussion.	July 26
Finalization of the interface specification (return codes).	August 4
Revised cluster security design document	Sept. 15
Stub library for the CtS interface	Sept. 30

2.4 NLS

Being part of the cluster project, CtS is NLS enabled. As part of this support, CtS ships a message catalog, ct_sec.cat, and a message header file, ct_sec_msg.h. During each call to ct_sec_get_error_msg() routine, CtS tries to open the message catalog. If successful, tries to retrieve the message.

The maximum length of the messages is limited to 1024 characters.

The message catalog shipped by CtS contains error messages for the return codes generated by the MALI routines. Each MPM will provide its own message catalog for its return codes.

2.5 Packaging

The cluster security software is packaged in a separate lpp from the cluster code. Coresquisites must be created between the security lpp and the cluster client lpp. Also, the cluster base lpp will prereq the security lpp.

TBD: need to discuss w/Sammuel Benjamin in more detail.

3.2 Cluster Environment

This chapter presents the security services environment and how they fit into the cluster picture. From the previous security discussions you should be familiar with concepts like network credential (identity) and security context, and why applications must acquire and use them (steps required to establish a secure communication will be described later in the document). But in order to acquire and use a network identity, applications must be able to determine several things, like what is the underlying security mechanism used in the cluster; or what identity are they going to use to acquire credentials; or what happens if a cluster system administrator changes the underlying security mechanism.

All these questions must find an answer in order to create a reliable system that can dynamically be changed to the administrator's preference. The intension of CtS is to provide an interface that is generic, meaning that clients of this interface do not deal directly with the complexity of and the differences between different security mechanisms. Clients program to only one interface and they are not impacted by the addition, in the future, of a new security mechanism to the cluster. The installation and configuration of the new underlying mechanism must follow strict guidance rules (to come in the future) for the cluster to work correctly.

3.2.1 Storing Configuration Information

The Cluster Security system design document specifies a single security mechanism in use at a time in the cluster. One can regard this specification as a restriction, however, a good one. CtS can determine, during initialization, what security mechanism is being in use, and can load and initialize the corresponding plugable module. Please read Section 5.0 Implementation of Mechanism Plugable Modules (MPMs) for more details on the design of the plugable modules.

To help CtS determine what security mechanism is configured and used in a cluster, this information must be stored in the System Registry. We can define a table in the System Registry that contains info about the underlying security mechanism: what mechanism, the time of configuration, the admin who configured it, and some other info that may be required/helpful.

Any user in the cluster (authenticated or not) should be allowed to read this information by calling an API routine and from the command line--we can offer a command that does that, for example, lssecinfo, with some flags...) and only the cluster administrator(s) have permission to change this information. The access control must be enforced by the System Registry server.

Issue 1: *Mechanism info table: structure and initialization.*

Need to determine what type of information is required in the mechanism table and how many entry we have to allow (taking into consideration that we support more than one security mechanism during migrations.)

Issue 2: *Recovery from System Registry/security mechanism failures.*

What happens when the System Registry is not functional? We, definitely, won't be able to retrieve security mechanism info. I guess if the System Registry is not functional, the entire cluster is not functional, and we have a bigger problem than reading the security mechanism info.

What if the System Registry is not functional because of a problem in the underlying security mechanism? This can be solved by starting the registry server in a service mode that doesn't use CtS. It is up to the System Registry team to determine if this is a viable idea and, if it is, to define the functionality in this mode.

Three requirements will be filed with the System Registry design component (SR_design) for the following:

1 start the registry without security enabled
2 use of Unix Domain Socket for registry communication
3 replication layer override for manual intervention

Another piece of information that must be stored is related to the MPMs. When a new plug-in is installed on a system, we must write some information about the plug-in (the name of the plug-in and the mechanism it corresponds to) in a local host configuration file, for example, /var/ct/<cluster_id>/security/ct_sec.cfg. This file should be used by CtS only. Everybody will be able to read this file and only root will be able to write to it.

3.2.2 Changing Security Configuration

The design of CtS provides flexibility to the cluster administrator in establishing the underlying security mechanism to be used, so long as there is a plugable module for that mechanism. The plug-in mechanism is introduced to allow CtS to dynamically load the module required for a security mechanism. The consequence is that the cluster administrator(s) should be able to install and configure a new security mechanism, change the underlying security mechanism info in the System Registry, and the cluster should continue to work correctly. There are some issues related to the change of the security mechanism, and some will not be answered in this design document (at least not completely), but in the install/config design document. However, CtS should provide the correct functionality to deal with these issues.

Let's assume we have a client-server application that uses the CtS API. Both the client and the server have network credentials for a configured security mechanism, and they established a security context. They are, indirectly, using the mechanism's libraries and runtime. Now, suppose an administrator changes the configuration for the mechanism in use to a new security mechanism. We assume, for simplicity, that the new mechanism is functional and configured correctly (i.e. all the required files are installed and all the principals/users have been created, etc.).

Issue 3: *Detect changes in the configuration of security mechanism.*

The question is at what point in time are the client and server supposed to detect a change in the security configuration? A sane answer would be to wait until CtS clients fail, at which point in time CtS could check if a change occurred, and if so, acquire network credential for the new security mechanism. The problem is that if the old security mechanism is still functional, the CtS client won't fail.

Let's take a more in-depth look at what is going on during a configuration change, and let's analyze the situations most affected by changes in security mechanisms.

3.2.2.1 Migrating To A New Security Mechanism

Under normal circumstances, cluster trusted services (which are CtS clients) have network credentials and they are able to accept/initiate security contexts from their clients/with other services. Of course, for this to happen, the underlying security mechanism must be correctly configured and in use (libraries and runtime).

CtS clients have one security token (that references the security services environment info: network credentials, network name, etc.) and several context tokens (that reference the security context info: client's network identity, client's PAC, etc.). These tokens point to opaque *blobs*, and CtS clients don't care what these *blobs* contain. From the CtS's point of view, the *blobs* are made of two parts: mechanism abstract information (the MAL level); and mechanism specific information (the MPM level). The latter is opaque to the MAL routines and strongly dependent on the security mechanism. This part is the heart of CtS and it is the most affected by a change in the security mechanism.

Issue 4: *What happens to established tokens when a configuration change occurs?*

The security service token should not be affected by a change in security mechanism because it deals with mechanism abstract information. The token descriptor contains a list of mechanism specific data, and some members of this list may be affected.

The context tokens will definitely be affected because they are security mechanism dependent.

The configuration changes may have different effects, depending on how the change is performed. Security mechanism changes are not that often (however, CtS must handle them), and I regard them as ***migrations***. A first condition for a successful migration is that the new security mechanism must be correctly installed and configured (I include here all the cluster specific configuration: creation of principals/users, keys/passwords, local files, etc.)

CtS intends to handle migrations in a seamless fashion, with certain limits, of course. These limits are imposed by the unconfiguration/removal of the old security mechanism. A migration from one security mechanism to another must be designed and implemented with care for the data/files that are already in use by the running CtS clients (otherwise unexpected results may occur, like memory violations, illegal instructions, etc.). As long as the old security mechanism is still functional, CtS clients are able to use the security services correctly.

The problem is that we don't want the CtS clients to continue using the old security mechanism, however functional it may be (the purpose of a migration is to use the new security mechanism). We want to detect a change in the CtS environment as soon as possible and take appropriate actions.

3.2.2.2 Detecting Security Mechanism Changes

The security tokens represent long-lived data. They should be created at program initialization and destroyed when the program finishes. In contrast, the context tokens are short-lived data. They are created when a request is coming from a client and destroyed when the client closes the connection.

The logical points of checking for changes in the security mechanism are before initiating security contexts (for the client) and before accepting security contexts (for the server). With other words, CtS checks for changes when the clients call `ct_sec_get_client_creds()` to acquire the process's credentials and initiate a security context with a server.

> **Note:** A CtS client can be a short-lived process (like a CLI command) or a long-lived process (a cluster service's daemon). The first one depends on the network identity and credential inherited from the invoker. The latter must do a `ct_sec_login_as_service()` in order to acquire network identity and credential. Both may need to initiate security contexts with servers or other services. The security token descriptor contains information to determine in what category the caller falls.

From the security token descriptor we can determine if the caller logged on to the network as a service, therefore being a daemon. In this case, CtS checks if the configured security mechanism (from the System Registry) is the same with what it is in the security token descriptor. If it is the same, CtS continues with initiating the security context. If it is a different security mechanism, CtS does the following:

- loads and initializes the plug-in for the new security mechanism. If it fails, returns error.
- determines the network name of the service for the new security mechanism (the network name may be different from one security mechanism to another).
- acquires network credentials for the daemon using the new security mechanism. If it fails, returns error.
- initializes a security context with the target service. If it fails, returns error.

In case of a server (service daemon), CtS checks for changes in security mechanism when it calls `ct_sec_authenticate_client()`. The server calls this routine when it needs to process the client's request to accept a security context. The client sends a context control data (CCD) buffer to the server, and the server passes it to this routine. We add the mechanism code to the CCD buffer to make it easier for CtS to determine if the client and server share the same mechanism. If the mechanism from the CCD buffer is the same with the security mechanism used by the server, then we are lucky. If not, CtS checks if the mechanism coming from client is the same with the one in the System Registry. If it isn't, returns error. If it is, CtS does the following:

- loads and initializes the plug-in for the new security mechanism. If it fails, returns error.
- acquires network credentials for the new mechanism.
- processes the client's request to accept the security context.

The description above of what is happening during a migration does not discuss synchronization. The security token descriptor may be used by several threads to create context token descriptors for connections coming from different clients at the same time. If one client is using the new security mechanism, it doesn't mean that all the others are using it. CtS must provide a single write/multiple read type of synchronization to the data that is affected by changes in security mechanism configuration (most probably the mechanism specific info only).

3.2.2.3 Handling Security Mechanism Errors

The design of CtS provides support for multiple security mechanisms at a time. This does not mean that we implement a negotiation protocol between a client and a server to determine what security mechanism will be used. The support of multiple security mechanism is only for migration purposes.

When CtS acquires a new network identity and credential (for the new security mechanism), chances are that there are some security contexts for the old mechanism. They are still good so long as the old security mechanism is functional. Problems occurs when some other actions are taken, like verify permission, or whatever operation that involves the security token (which has credential for the new mechanism). Whenever an MPM routine returns a security mechanism error, CtS will check if there are any changes in the security mechanism, and they will be reported back to the caller. In case of a context token related error, the CtS client must establish a new connection with the server, using the new network identity and credential.

3.2.2.4 Unpredictable Behavior

Until now we talked about migration under the assumption that the administrator keeps the old security mechanism functional for a while (at least until all security contexts expires). What happens if the administrator unconfigures and removes the old security mechanism? CtS clients may simply crash or hang, leaving the cluster in a unusable state. I guess the best answer is "the behavior is unpredictable."

3.5 Daemon to Daemon Communication

Daemon to daemon communication refers to the communication between two instances of the same service. In cluster, all instances of a service share the same network identity and password/key. Each instance acquires network identity during process initialization which is constantly renewed using the password/key. The synchronization of the keys in cluster is performed by the Key Management component of cluster security.

Because all instances of a daemon have the same network indentity, the communication between them can be simplified. There are two approaches: (1) authentication-based communication; and (2) non-authentication-based communication. Let's discuss the two approaches in more detial.

3.5.1 Authentication-Based Communication

This approach is characterized by the fact that an instance of a daemon (initiator) that needs to communicate with one or more instances of the same daemon (acceptors) must authenticate to only one other instance. The service ticket that the initiating instance obtains from the security mechanism is valid for all the accepting instances because they share the same network identity. The initiating instance sends the service ticket to the other instances, authenticating to them. If mutual authentication is required, each accepting instance will send back a context control buffer (CCD). The initiator must authenticate only the first accepting instance and compare this CCD buffer with the CCD buffers received from the rest of the accepting instances.

After authentication, the same session key is shared between the initiating instance and the accepting instances, and , therefore, encrypted/signed messages can be broadcasted.

3.5.2 Non-Authentication-Based Communication

This approach differs from the previous one because the initiating instance does not authenticates to the accepting instances. All instances already have a network identity and they share the same key. The initiating instance can use the shared key to encrytp/sign messages that are broadcasted to the accepting instances. Each accepting instance can then decrypt/verify the message received, and if successful, it can guarantee that it was received from somebody that knows the key they share.

This approach implies new routines that CtS must provide to encrypt/decrypt and sign/verify messages using a key versus a security context.

Issue 8: *What daemon-to-daemon communication approach should be used in the cluster?*

The correct answer to this issue is that tcp based daemons should use the authentication-based communication approach for authenticating different instances, and the udp based daemon should use the non-authentication communication approach.

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- □ BLACK BORDERS
- □ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- □ FADED TEXT OR DRAWING
- □ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- □ SKEWED/SLANTED IMAGES
- □ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- □ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- □ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- □ OTHER: ______________________

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.